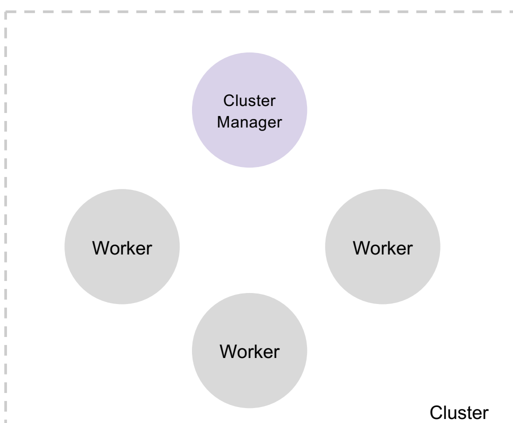
Batch Processing

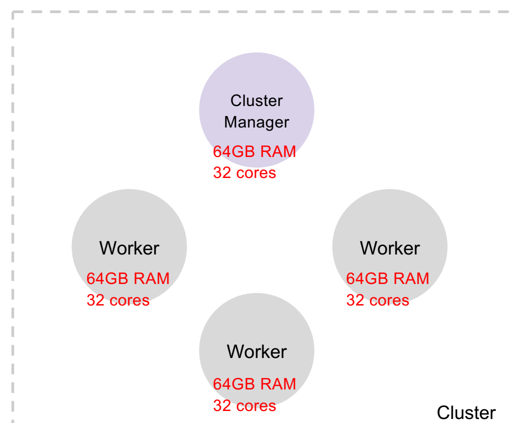
COS 518: *Distributed Systems*
Lecture 11

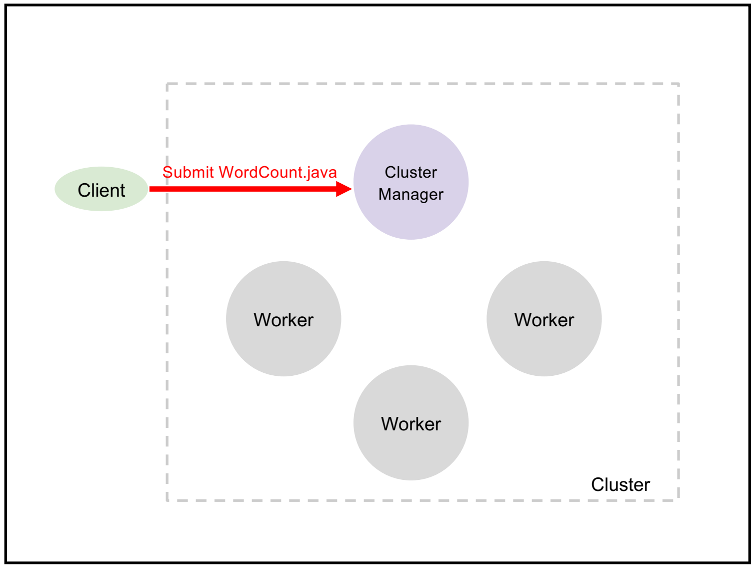
Mike Freedman

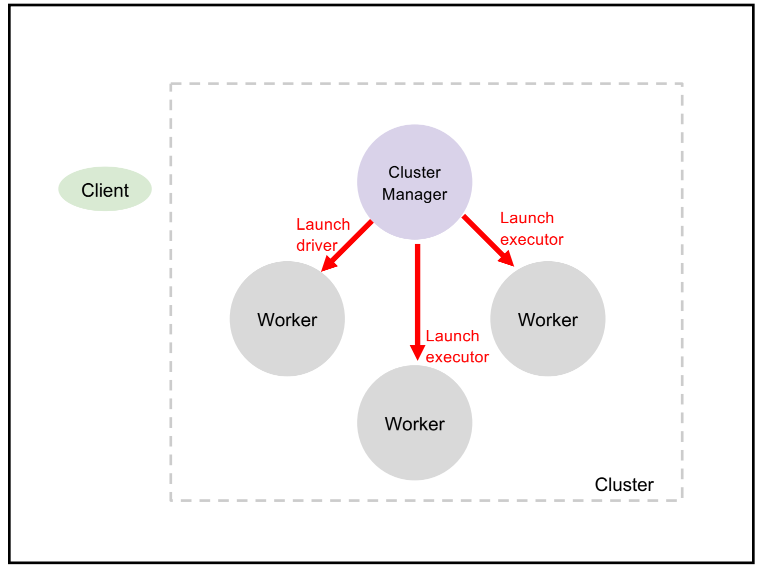
Basic architecture

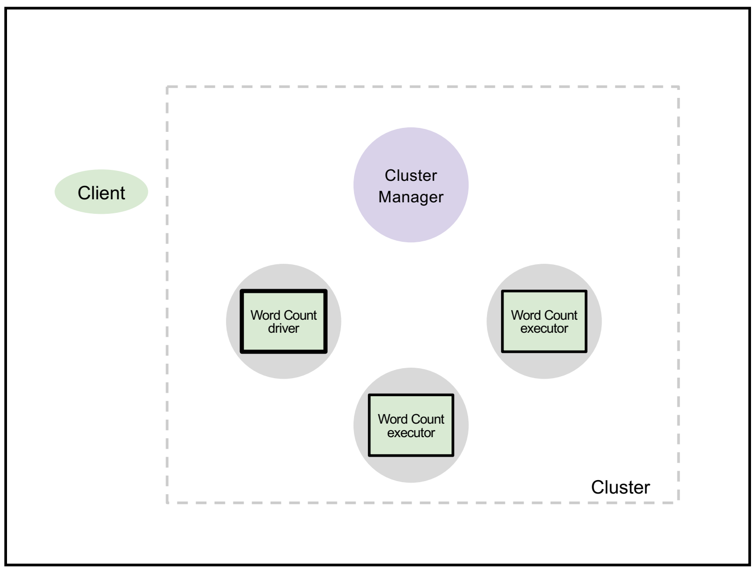
in “big data” systems

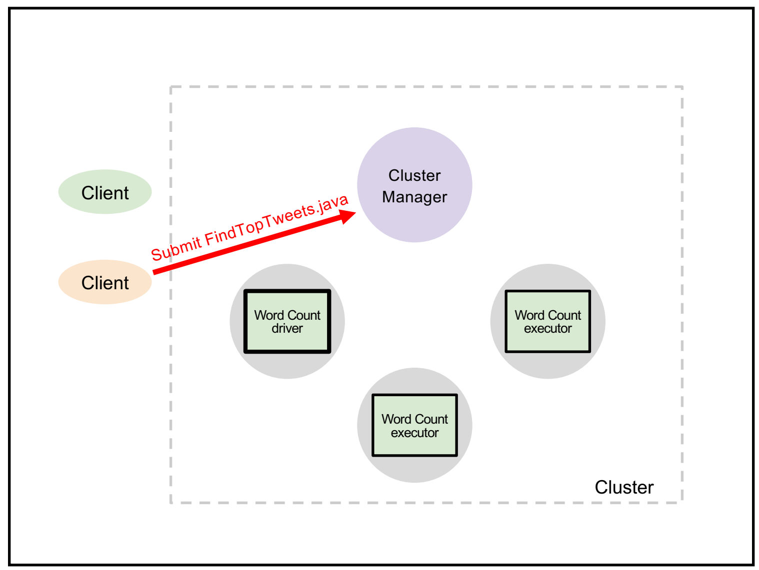


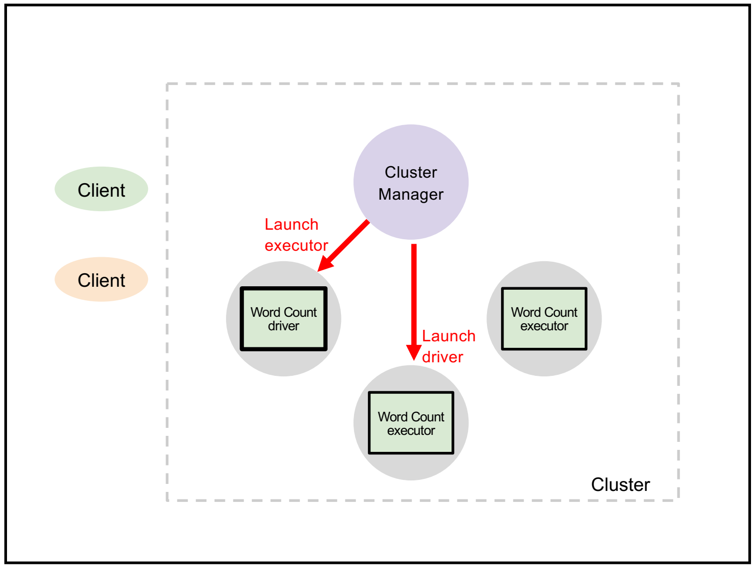


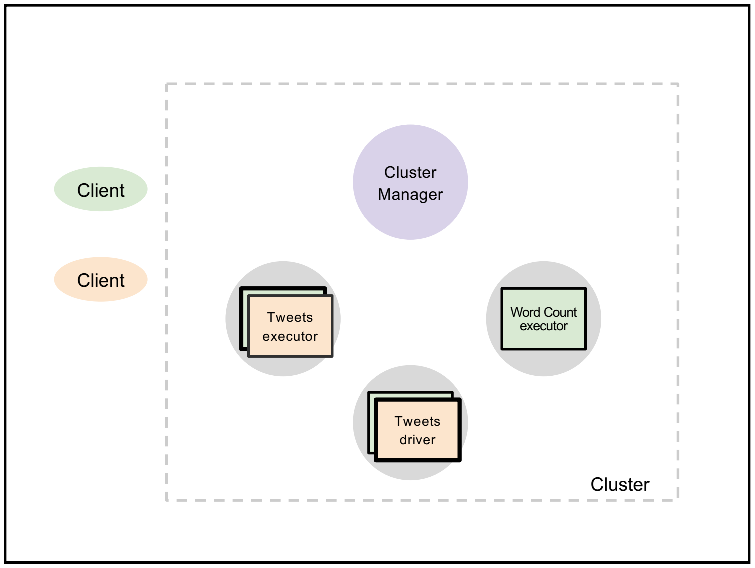

Client
Submit WordCount.java
Cluster Manager
Worker
Worker
Worker
Cluster

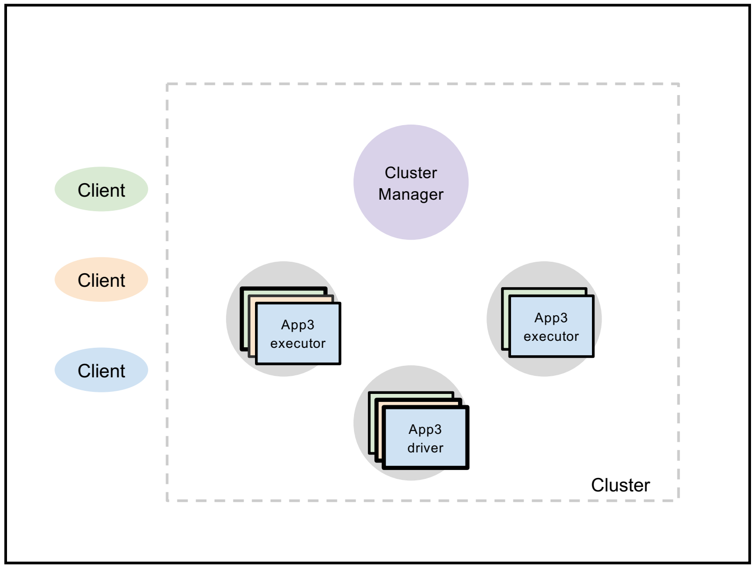

Client
Cluster Manager
Launch driver
Launch executor
Launch executor
Worker
Worker
Worker
Cluster


Client
Cluster Manager
Word Count driver
Word Count executor
Word Count executor
Cluster


Client
Client
Submit FindTopTweets.java
Cluster Manager
Word Count driver
Word Count executor
Word Count executor
Cluster







Basic architecture

Clients submit applications to the cluster manager

Cluster manager assigns cluster resources to applications

Each Worker launches containers for each application

Driver containers run main method of user program

Executor containers run actual computation

Examples of cluster manager: YARN, Mesos

Examples of computing frameworks: Hadoop MapReduce, Spark

12

Two levels of scheduling

Cluster-level: Cluster manager assigns resources to applications

Application-level: Driver assigns *tasks* to run on executors

- A task is a unit of execution that operates on one *partition*

Some advantages:

- Applications need not be concerned with resource fairness
- Cluster manager need not be concerned with individual tasks
- Easy to implement priorities and preemption

13

Case Study: MapReduce

(Data-parallel programming at scale)

14

Application: Word count

Hello my love. I love you, my dear. Goodbye.

↓

hello: 1, my: 2, love: 2, i: 1, dear: 1, goodbye: 1

15

Application: Word count

Locally: tokenize and put words in a hash map

How do you parallelize this?

- Split document by half
- Build two hash maps, one for each half
- Merge the two hash maps (by key)

16

How do you do this in a distributed environment?

When in the Course of human events, it becomes necessary for one people to dissolve the political bands which have connected them with another, and to assume, among the Powers of the earth, the separate and equal station to which the Laws of Nature and of Nature's God entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the separation.

Input document

When in the Course of human events, it becomes necessary for one people to

dissolve the political bands which have connected them with another, and to assume,

among the Powers of the earth, the separate and equal station to which the Laws of

Nature and of Nature's God entitle them, a decent respect to the opinions of mankind

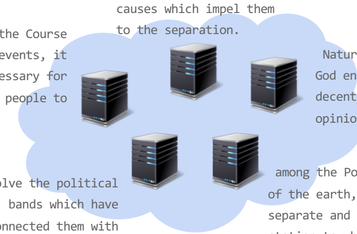
requires that they should declare the causes which impel them to the separation.

Partition

requires that they should declare the causes which impel them to the separation.

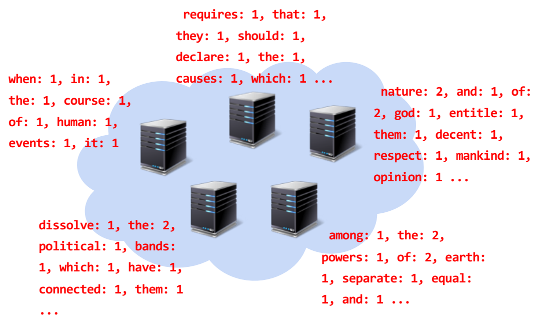
When in the Course of human events, it becomes necessary for one people to

Nature and of Nature's God entitle them, a decent respect to the opinions of mankind

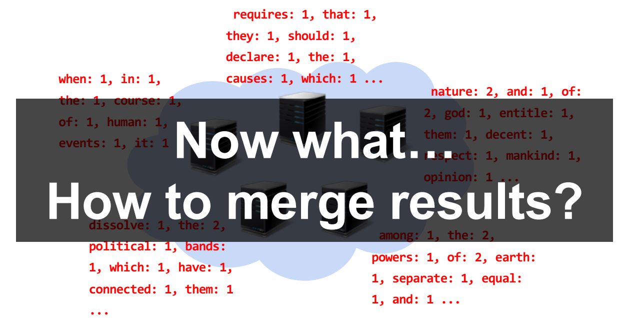


dissolve the political bands which have connected them with another, and to assume,

among the Powers of the earth, the separate and equal station to which the Laws of



Compute word counts locally



Compute word counts locally

Merging results computed locally

Several options

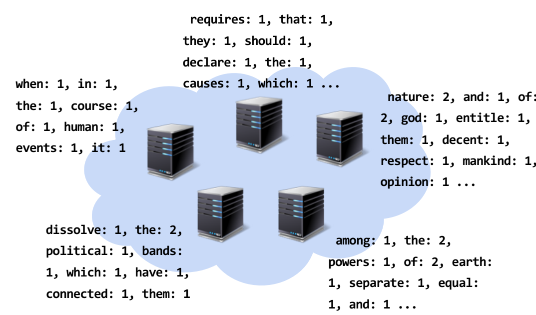
Don't merge — requires additional computation for correct results

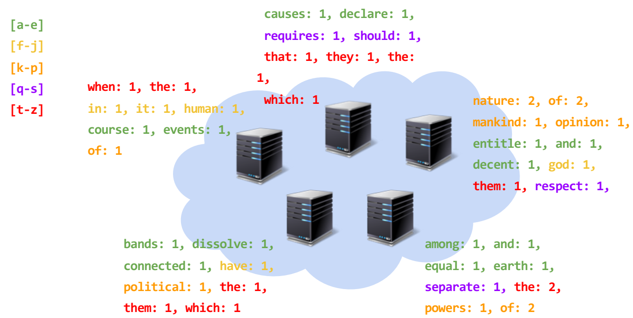
Send everything to one node — what if data is too big? Too slow…

Partition key space among nodes in cluster (e.g. [a-e], [f-j], [k-p] ...)

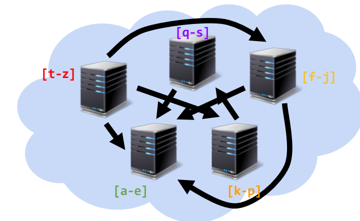
1. Assign a key space to each node
2. Partition local results by the key spaces
3. Fetch and merge results that correspond to the node's key space

23

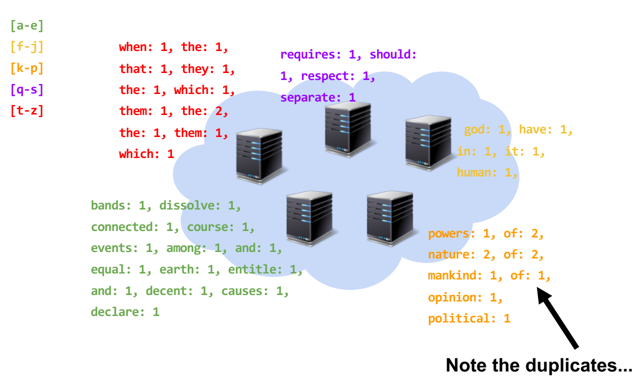


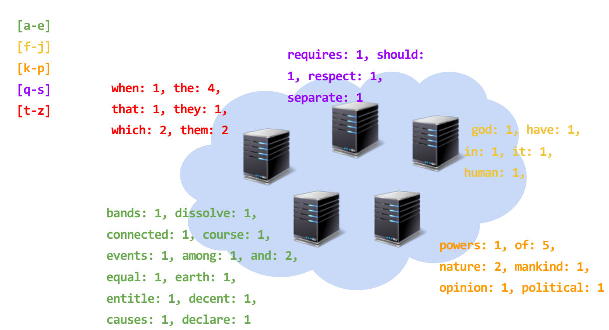


Split local results by key space



All-to-all shuffle





Merge results received from other nodes

MapReduce

Partition dataset into many chunks

Map stage: Each node processes one or more chunks locally

Reduce stage: Each node fetches and merges partial results from all other nodes

MapReduce Interface

`map(key, value) -> list(<k', v'>)`

Apply function to (key, value) pair

Outputs set of intermediate pairs

`reduce(key, list<value>) -> <k', v'>`

Applies aggregation function to values

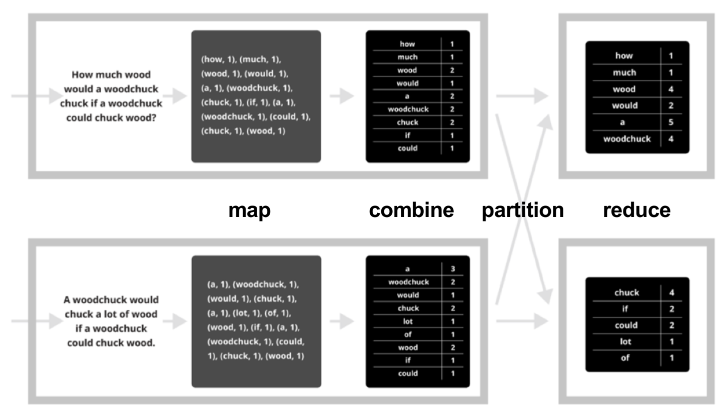
Outputs result

MapReduce: Word count

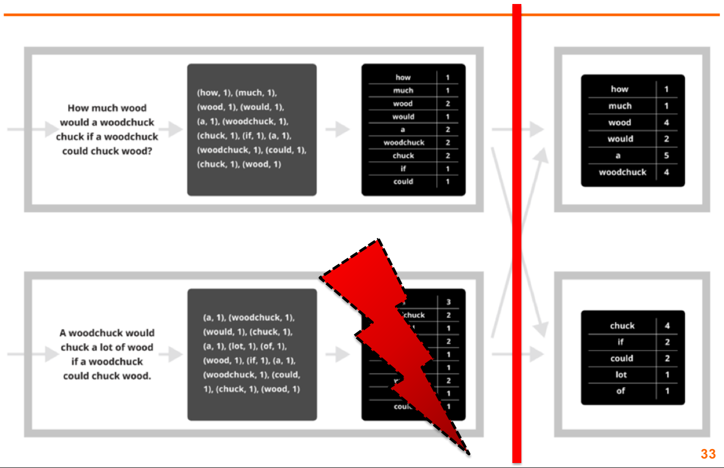
```
map(key, value):
  // key = document name
  // value = document contents
  for each word w in value:
    emit (w, 1)

reduce(key, values):
  // key = the word
  // values = number of occurrences of that word
  count = sum(values)
  emit (key, count)
```

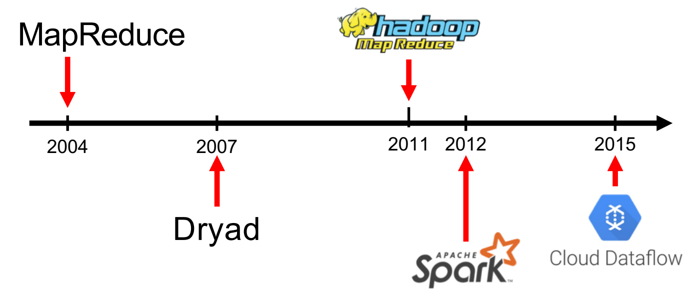

MapReduce: Word count



Synchronization Barrier



33



Brainstorm: Top K

Find the largest K values from a set of numbers

How would you express this as a distributed application?

In particular, what would map and reduce phases look like?

Hint: use a heap…

35

Brainstorm: Top K

Assuming that a set of K integers fit in memory…

Key idea...

Map phase: everyone maintains a heap of K elements

Reduce phase: merge the heaps until you're left with one

36

Brainstorm: Top K

Problem: What are the keys and values here?

No notion of key here, just assign the same key to all the values (e.g. key = 1)

Map task 1: [10, 5, 3, 700, 18, 4] → (1, heap(700, 18, 10))

Map task 2: [16, 4, 523, 100, 88] → (1, heap(523, 100, 88))

Map task 3: [3, 3, 3, 3, 300, 3] → (1, heap(300, 3, 3))

Map task 4: [8, 15, 20015, 89] → (1, heap(20015, 89, 15))

Then all the heaps will go to a single reducer responsible for the key 1

This works, but clearly not scalable…

37

Brainstorm: Top K

Idea: Use X different keys to balance load (e.g. X = 2 here)

Map task 1: [10, 5, 3, 700, 18, 4] → (1, heap(700, 18, 10))

Map task 2: [16, 4, 523, 100, 88] → (1, heap(523, 100, 88))

Map task 3: [3, 3, 3, 3, 300, 3] → (2, heap(300, 3, 3))

Map task 4: [8, 15, 20015, 89] → (2, heap(20015, 89, 15))

Then all the heaps will (hopefully) go to X different reducers

Rinse and repeat (*what's the runtime complexity?*)

38

Monday 3/25

Stream processing

39

Application: Word Count

```
SELECT count(word) FROM data
   GROUP BY word

cat data.txt
   | tr -s '[[:punct:][:space:]]' '\n'
   | sort | uniq -c
```

40

Using partial aggregation

1. Compute word counts from individual files
2. Then merge intermediate output
3. Compute word count on merged outputs

41

Using partial aggregation

1. In parallel, send to worker:
   – Compute word counts from individual files
   – Collect result, wait until all finished
2. Then merge intermediate output
3. Compute word count on merged intermediates

42

MapReduce: Programming Interface

```
map(key, value) -> list(<k', v'>)
```

– Apply function to (key, value) pair and produces set of intermediate pairs

```
reduce(key, list<value>) -> <k', v'>
```

– Applies aggregation function to values

– Outputs result

43

MapReduce: Programming Interface

```
map(key, value):
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(key, list(values):
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

44

MapReduce: Optimizations

`combine(list<key, value>) -> list<k,v>`

– Perform partial aggregation on mapper node:

<the, 1>, <the, 1>, <the, 1> → <the, 3>

– reduce() should be commutative and associative

`partition(key, int) -> int`

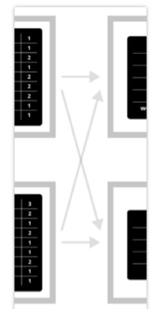
– Need to aggregate intermediate vals with same key

– Given n partitions, map key to partition $0 \leq i < n$

– Typically via hash(key) mod n

45

Fault Tolerance in MapReduce



- Map worker writes intermediate output to local disk, separated by partitioning. Once completed, tells master node.
- Reduce worker told of location of map task outputs, pulls their partition's data from each mapper, execute function across data
- Note:
  - "All-to-all" shuffle b/w mappers and reducers
  - Written to disk ("materialized") b/w *each* stage

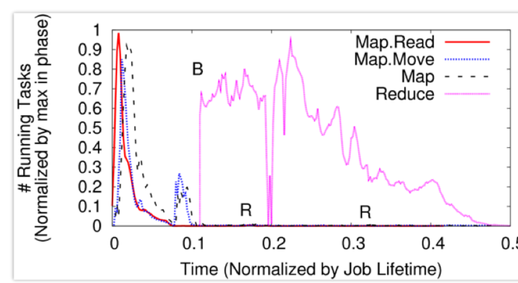
46

Fault Tolerance in MapReduce

- Master node monitors state of system
  - If master failures, job aborts and client notified
- Map worker failure
  - Both in-progress/completed tasks marked as idle
  - Reduce workers notified when map task is re-executed on another map worker
- Reducer worker failure
  - In-progress tasks are reset to idle (and re-executed)
  - Completed tasks had been written to global file system

47

Straggler Mitigation in MapReduce



- Tail latency means some workers finish late
- For slow map tasks, execute in parallel on second map worker as "backup", race to complete task

48